WELCOME to the **NEW and UPGRADED dssRookie** website! (March 7, 2012)

We have upgraded to a faster SERVER ad upgraded all components of dssRookie. There's still some growing pains. Any errors, bugs, or suggestions, please make a post or notify any of the staff.

Site may be off and on-line intermittently for a few weeks while we massage it into shape!

Thank you all for your patience!
---

dssRookie is now giving VIP Members support on **IKS Servers**!

You can login to the IKS forums with a VIP Membership!

We will be providing **GUIDES** to setting up of your various receivers, Dreabox, Dreamlink, NeoSat and others.
We will be posting **FILES** for your particular receiver to get it ready for FTA, IKS, etc.
We will be **DISCUSSING** various different IKS providers, channel lists, LNB settings, etc.
We will be providing **HELP** in setting up your receivers, etc.

Bigger, Faster, Better with lots more options for VIP Members...

To get full IKS support join dssRookie.. Details can be found in **Join and be a VIP Member**

If this is your first visit, be sure to check out the **FAQ** by clicking the link above. You may have to **register** before you can post: click the register link above to proceed. To start viewing messages, select the forum that you want to visit from the selection below.

Results 1 to 10 of 37 Page 1 of 4 1 2 3 4 Last

Thread: STi5107 Disassembly

Thread Tools Display

09-26-2010, 02:10 PM #1

jccl2
Member

Join Date: Jun 2009
Posts: 10
Thanks: 0
Thanked 0 Times in 0 Posts

STi5107 Disassembly

Hello,

I need help to disassembly STi5107 on IDA Pro, tried to ran on emulator but don´t know where the bootloader offset. Used 0x7FFFFFFE like on Sti5517 but on hex it´s had only FF FF FF .... Any tips how to run on emulator and address on IDA Pro or any other ?

CPU STi5107
Memory HY5DU561622FTP-5
Flash S29GL064N90FFI020

DUMP
h**p://www.mediafire.com/?uqg5y287lntimxn

Reply With Quote

09-27-2010, 12:52 PM #2

jvvh5897
VIP Coders Corner

Join Date: Apr 2005
Posts: 1,106
Thanks: 0
Thanked 43 Times in 32 Posts

The way I would start is to google "STi5107"--when I did that I found a number of threads about jtagging--one had a service manual for a similar processor and there might be some good info in that for you. Read a bunch of the hits that google comes up with. You might find info about if the chip is ST20 or ST40 code, DC2 or DC3 (I saw ST20 and DC3 for your chip).

Next thing is to LOOK at the dump file you posted a link to--use a hexviewer or hexeditor, there usually is some indication of where it is going to be in memory. ST uses pointers to information and you can usually find those pointers.

Reply With Quote

09-28-2010, 11:43 AM #3

jccl2
Member

Join Date: Jun 2009
Posts: 10
Thanks: 0
Thanked 0 Times in 0 Posts

Thanks for the reply.

I had this service guide, this had the pinouts for 5105.
I used the HexEditor to see .bin but, really don't know how to check where the information going to memory. I'm really a noob..... Tried to use st20disassembly they generated .asm but I think they put the info on wrong address. I'll try to check if can found the boot address on this dump.

Thanks.

Reply With Quote

09-28-2010, 01:16 PM #4

jvvh5897
VIP Coders Corner

Join Date: Apr 2005
Posts: 1,106
Thanks: 0
Thanked 43 Times in 32 Posts

File size is 0x8000ff--so you likely remove the first 0x100 bytes to get what is written to flash. But maybe not--the code at the file start looks to be loading the pointer 0x40000604 and doing a call to there as the entry point. And at 0x604 into the file you do see a load of address 0x40000d70 and a call to there. All that suggests that the real entry point is at the first byte of the file and it should be at 0x40000000.
Search for "run" finds me "runtime c1rtl"--so code is ST20C1 machine code--so be sure not to use ST20C2 or C4.
Search for 0x00000080 finds a table that starts around 1c7e8 in the file that might be a section table (note at 0x614 you find the pointer to 0x4001c7e8 and at 0x610 a pointer to boot data 0x4001d040 table, you also see the address 0x80000400 and that is likely the starting static_link)--writes zeros starting at c0800000, so RAM base address is 0xc0000000. You also see moves of code to RAM from flash adresses that look like 0x40000f64--so flash base address is 0x40000000.
Search for "boot" found me some interesting strings early in the code--one of the most interesting is "Coship N5166HSG BOOTER"--so looks like the code is for a Coship box. Looks to me like the first 0x20000 bytes is a boot. And code after 0x20000 is app--just a guess though, though you see a big gap after 0x85e5 to the next hit on "boot".

Reply With Quote

09-28-2010, 11:55 PM #5

jccl2
Member

Join Date: Jun 2009
Posts: 10
Thanks: 0
Thanked 0 Times in 0 Posts

Many thanks for the explain. Now the doubs....
Tried to put this values on IDA
RAM: 0xC0000000 / Size: 0x0F801F00 ???
ROM: 0x40000000 / Size: 0x00800100
Loading Address: 0x40000000

But nothing send to the RAM address.

Tried ran on st20sim but w/o luck.....

Thanks.

Reply With Quote

09-29-2010, 02:10 PM #6

jvvh5897
VIP Coders Corner

Join Date: Apr 2005
Posts: 1,106
Thanks: 0
Thanked 43 Times in 32 Posts

I always looked at the section table and figured out what was moved to RAM--if code, then I created a file with code in place and disassembled that. Never played with emulation in IDA, don't know if it works for ST20 code of any type let alone for your particular file. The newer processors

that use ST20C1 code are really using a set called "enhanced C1" I don't know if IDA has the full instruction set for that, it may just have the old C1 set and so could be missing a number of instructions. The few times I looked at C1 code with the version of IDA I have, I saw a number of instructions not being disassembled correctly (think they were something like register push and pops as they usually were at the start and end of a routine and C1 code does not push registers onto stack with a call the way that C2 code does).

You could write an IDC to read the section table and use it to create a RAM image. Or you could use a C compiler to do it, or qbasic, or any other programming language that you like that can manipulate files and read bytes. I had some luck with lcc-win32 free C compiler and a source code called hexv to read from files, I first wrote in qbasic to extract section table and put in readable form for my eyes, but to be able to write anything I first had to LOOK at the table and see what it was doing. You might even be able to use st20osf to run the code that uses the section table--though st20osf (and the sourceforge site st20emu source code that it is based on) is only for st20c2\c4 code, st20c1 has many instructions that do work the same as c2 (ldc, ldl\stl, ldnl\stnl, ldnlp\stnlp, and adj are all the same as I recall), it might work or you might be able to mod the source code to do it well enough (you can compile st20emu with lcc-win32).

Reply With Quote

09-29-2010, 09:22 PM #7

jccl2
Member

Join Date: Jun 2009
Posts: 10
Thanks: 0
Thanked 0 Times in 0 Posts

"I always looked at the section table and figured out what was moved to RAM--if code, then I created a file with code in place and disassembled that."
It´s exacted what I need, but don´t know where are the "section table" and how to indentify =(.I supose the part of the dump are encrypted and this part were uncrypted on memory. They are on offset 1fc to 3f2.

Now I tried to use JASMA, DASMST20 and st20osf2.

For DASMst20 I can use entry point 0x40000604 or 0x40000000 ?

Thanks.

Last edited by jccl2; 09-30-2010 at 12:10 PM. ***Reason:*** *More info.....*

Reply With Quote

09-30-2010, 03:01 PM #8

jvvh5897
VIP Coders Corner

Join Date: Apr 2005
Posts: 1,106
Thanks: 0
Thanked 43 Times in 32 Posts

DASMST20 and st20osf2 are of little use as they are for ST20C2/4 code and not forST20C1.

I ran the section table (location noted in earlier post) through my old qbasic program to put it in more easily read format, as usual it is number of bytes, source location, destination address and if source address is 0x80000000 then you put zeros in the destination addres for the number of bytes listed.

```
00000004 ; 80000000 ; C0080000 ; 00000004 ; 80000000 ; C0200000
000067A4 ; 40000F64 ; C1DEFC00 ; 00000010 ; 80000000 ; C1DF63A4
00011760 ; 40007708 ; C1DF63B4 ; 0000001C ; 80000000 ; C1E07B14
0000000C ; 40018E68 ; C1E07B30 ; 00000034 ; 80000000 ; C1E07B3C
00000120 ; 40018E74 ; C1E07B70 ; 00000018 ; 80000000 ; C1E07C90
0000004C ; 40018F94 ; C1E07CA8 ; 00000014 ; 80000000 ; C1E07CF4
00000038 ; 40018FE0 ; C1E07D08 ; 00000010 ; 80000000 ; C1E07D40
00000100 ; 40019018 ; C1E07D50 ; 00000028 ; 80000000 ; C1E07E50
000000D4 ; 40019118 ; C1E07E78 ; 00000010 ; 80000000 ; C1E07F4C
000000A8 ; 400191EC ; C1E07F5C ; 0001900C ; 80000000 ; C1E08004
```

000000C4 ; 40019294 ; C1E21010 ; 0000001C ; 80000000 ; C1E210D4
00000020 ; 40019358 ; C1E210F0 ; 00000014 ; 80000000 ; C1E21110
00000054 ; 40019378 ; C1E21124 ; 00000010 ; 80000000 ; C1E21178
0000002C ; 400193CC ; C1E21188 ; 00000018 ; 80000000 ; C1E211B4
00000034 ; 400193F8 ; C1E211CC ; 00000010 ; 80000000 ; C1E21200
00000038 ; 4001942C ; C1E21210 ; 00000018 ; 80000000 ; C1E21248
00000018 ; 40019464 ; C1E21260 ; 0000001C ; 80000000 ; C1E21278
00000018 ; 4001947C ; C1E21294 ; 00000018 ; 80000000 ; C1E212AC
00000090 ; 40019494 ; C1E212C4 ; 00000010 ; 80000000 ; C1E21354
00000020 ; 40019524 ; C1E21364 ; 00000010 ; 80000000 ; C1E21384
00000020 ; 40019544 ; C1E21394 ; 00000010 ; 80000000 ; C1E213B4
00000020 ; 40019564 ; C1E213C4 ; 00000010 ; 80000000 ; C1E213E4
00000020 ; 40019584 ; C1E213F4 ; 00000010 ; 80000000 ; C1E21414
00000020 ; 400195A4 ; C1E21424 ; 00000010 ; 80000000 ; C1E21444
00000020 ; 400195C4 ; C1E21454 ; 00000010 ; 80000000 ; C1E21474
00000020 ; 400195E4 ; C1E21484 ; 00000010 ; 80000000 ; C1E214A4
0000004C ; 40019604 ; C1E214B4 ; 0000001C ; 80000000 ; C1E21500
00000010 ; 40019650 ; C1E2151C ; 0000001C ; 80000000 ; C1E2152C
00000018 ; 40019660 ; C1E21548 ; 00000014 ; 80000000 ; C1E21560
00000020 ; 40019678 ; C1E21574 ; 00000014 ; 80000000 ; C1E21594
00000064 ; 40019698 ; C1E215A8 ; 00000010 ; 80000000 ; C1E2160C
00000020 ; 400196FC ; C1E2161C ; 00000010 ; 80000000 ; C1E2163C
00000020 ; 4001971C ; C1E2164C ; 00000010 ; 80000000 ; C1E2166C
00000020 ; 4001973C ; C1E2167C ; 00000010 ; 80000000 ; C1E2169C
00000020 ; 4001975C ; C1E216AC ; 00000010 ; 80000000 ; C1E216CC
00000020 ; 4001977C ; C1E216DC ; 00000010 ; 80000000 ; C1E216FC
00000020 ; 4001979C ; C1E2170C ; 00000010 ; 80000000 ; C1E2172C
00000068 ; 400197BC ; C1E2173C ; 00000010 ; 80000000 ; C1E217A4
00000020 ; 40019824 ; C1E217B4 ; 00000010 ; 80000000 ; C1E217D4
00000020 ; 40019844 ; C1E217E4 ; 00000010 ; 80000000 ; C1E21804
00000020 ; 40019864 ; C1E21814 ; 00000010 ; 80000000 ; C1E21834
00000020 ; 40019884 ; C1E21844 ; 00000010 ; 80000000 ; C1E21864
00000020 ; 400198A4 ; C1E21874 ; 00000010 ; 80000000 ; C1E21894
00000020 ; 400198C4 ; C1E218A4 ; 00000010 ; 80000000 ; C1E218C4
00000020 ; 400198E4 ; C1E218D4 ; 00000010 ; 80000000 ; C1E218F4
00000020 ; 40019904 ; C1E21904 ; 00000010 ; 80000000 ; C1E21924
00000020 ; 40019924 ; C1E21934 ; 00000010 ; 80000000 ; C1E21954
00000020 ; 40019944 ; C1E21964 ; 00000010 ; 80000000 ; C1E21984
00000020 ; 40019964 ; C1E21994 ; 00000010 ; 80000000 ; C1E219B4
00000020 ; 40019984 ; C1E219C4 ; 00000014 ; 80000000 ; C1E219E4
00000030 ; 400199A4 ; C1E219F8 ; 00000014 ; 80000000 ; C1E21A28
00000020 ; 400199D4 ; C1E21A3C ; 00000010 ; 80000000 ; C1E21A5C
00000020 ; 400199F4 ; C1E21A6C ; 00000010 ; 80000000 ; C1E21A8C
0000007C ; 40019A14 ; C1E21A9C ; 00000034 ; 80000000 ; C1E21B18
00000010 ; 40019A90 ; C1E21B4C ; 00000020 ; 80000000 ; C1E21B5C
00000018 ; 40019AA0 ; C1E21B7C ; 00000010 ; 80000000 ; C1E21B94
00000020 ; 40019AB8 ; C1E21BA4 ; 00000018 ; 80000000 ; C1E21BC4
00000018 ; 40019AD8 ; C1E21BDC ; 00000010 ; 80000000 ; C1E21BF4
00000020 ; 40019AF0 ; C1E21C04 ; 00000018 ; 80000000 ; C1E21C24
00000034 ; 40019B10 ; C1E21C3C ; 00000014 ; 80000000 ; C1E21C70
000000C4 ; 40019B44 ; C1E21C84 ; 00000014 ; 80000000 ; C1E21D48
00000018 ; 40019C08 ; C1E21D5C ; 00000018 ; 80000000 ; C1E21D74
00000018 ; 40019C20 ; C1E21D8C ; 00000018 ; 80000000 ; C1E21DA4
00000018 ; 40019C38 ; C1E21DBC ; 00000018 ; 80000000 ; C1E21DD4
00000020 ; 40019C50 ; C1E21DEC ; 00000010 ; 80000000 ; C1E21E0C
00000034 ; 40019C70 ; C1E21E1C ; 00000014 ; 80000000 ; C1E21E50
00000034 ; 40019CA4 ; C1E21E64 ; 00000014 ; 80000000 ; C1E21E98
00000018 ; 40019CD8 ; C1E21EAC ; 00000018 ; 80000000 ; C1E21EC4
00000018 ; 40019CF0 ; C1E21EDC ; 00000018 ; 80000000 ; C1E21EF4
00000018 ; 40019D08 ; C1E21F0C ; 00000018 ; 80000000 ; C1E21F24
00000018 ; 40019D20 ; C1E21F3C ; 00000018 ; 80000000 ; C1E21F54
00000020 ; 40019D38 ; C1E21F6C ; 00000010 ; 80000000 ; C1E21F8C
00000068 ; 40019D58 ; C1E21F9C ; 00000014 ; 80000000 ; C1E22004
00000008 ; 40019DC0 ; C1E22018 ; 00000014 ; 80000000 ; C1E22020
00000114 ; 40019DC8 ; C1E22034 ; 00000010 ; 80000000 ; C1E22148
000001F8 ; 40019EDC ; C1E22158 ; 00000054 ; 80000000 ; C1E22350
000001B4 ; 4001A0D4 ; C1E223A4 ; 00000044 ; 80000000 ; C1E22558
0000000C ; 4001A288 ; C1E2259C ; 00000030 ; 80000000 ; C1E225A8
00000010 ; 4001A294 ; C1E225D8 ; 00000404 ; 80000000 ; C1E225E8
00000004 ; 4001A2A4 ; C1E229EC ; 00000034 ; 80000000 ; C1E229F0
00000A40 ; 4001A2A8 ; C1E22A24 ; 0000007C ; 80000000 ; C1E23464

> 0000001C ; 4001ACE8 ; C1E234E0 ; 00000048 ; 80000000 ; C1E234FC
> 00000EC0 ; 4001AD04 ; C1E23544 ; 00000020 ; 80000000 ; C1E24404
> 000000E4 ; 4001BBC4 ; C1E24424 ; 00000010 ; 80000000 ; C1E24508
> 0000002C ; 4001BCA8 ; C1E24518 ; 00000038 ; 80000000 ; C1E24544
> 00000054 ; 4001BCD4 ; C1E2457C ; 00000010 ; 80000000 ; C1E245D0
> 00000050 ; 4001BD28 ; C1E245E0 ; 00024C30 ; 80000000 ; C1E24630
> 00000A4C ; 4001BD78 ; C1E49260 ; 0000003C ; 80000000 ; C1E49CAC
> 00000024 ; 4001C7C4 ; C1E49CE8 ; 00000000 ; 00000000 ; 00000000
> 00000000 ;

The entries of interest to you are:
000067A4 ; 40000F64 ; C1DEFC00 ; 00000010 ; 80000000 ; C1DF63A4
00011760 ; 40007708 ; C1DF63B4
there are two big chunks of the boot code moved to RAM with 0x10 bytes of zeros beween them. You should be able to use a hexeditor like XVI32 (free) to edit them out and paste into a file that you can disassemble with IDA Pro. Since there are only 0xf64 bytes of your boot that are executed in flash, you might want to disassemble that on its own so that you get the entry points to the RAM executed code. That c1df63b4 address implies that you have at least 32 MByte of RAM.

BTW, I looked at the stuff above 0x20000 a little and I'm tempted to say that it is not clear code. Either it is compressed or scrambled in some way--you might look in the boot code for strings like "inflate" or just "compress" to see if you can spot what is being done. The stuff above 0x20000 in the file seem to be in a number of parts and the stuff right around 0x20000 might be a scramble table for the stuff around 0x90000--still guessing though.

Reply With Quote

10-01-2010, 12:56 AM #9

jccl2
Member

Join Date: Jun 2009
Posts: 10
Thanks: 0
Thanked 0 Times in 0 Posts

Now I understand the table..... and how to search.

On the part of hex edit, I understand need to copy the begining bytes "between" zeros, it´s correct ?

Using the entire file.....
I tried to ran on IDA with:
RAM: 0xC0000000 / Size: 0x02000400
ROM: 0x40000000 / Size: 0x00800100
Loading Add: 0x40000000
Then pressed "C" on 0x40000F64 but they don´t sent anything to RAM: C1DEFC00 or to any RAM address. I sure are things wrong..... =/ and where I can set 0x80000000 address ?

And searching for Decompress, I find some on hexedit.

Many Thanks.

Reply With Quote

10-01-2010, 01:03 PM #10

jvvh5897
VIP Coders Corner

Join Date: Apr 2005
Posts: 1,106
Thanks: 0
Thanked 43 Times in 32 Posts

> Then pressed "C"

Pressing C only starts disassembly, disassembly does not emulate code or execute it in any way, it only converts machine code to assembly code.

> Then pressed "C" on 0x40000F64

But as the section table shows, the code at that address is moved to RAM, so you should not be trying to disassemble the code at the flash addr, you should be using the RAM addr.

> and where I can set 0x80000000 address

You don't set that addr anywhere in IDA.

You seem to think that IDA can emulate and execute ST20 code and I'm pretty sure that it can not. Course I don't know what version of IDA you have, maybe there are new versions that have that function. You can write IDC code for IDA that add functions, for instance, you could write code to read off the section table and move/clear as needed to create a RAM image of the boot and start disassembly. You can write IDC to find the start of routines with ajw commands and disassemble the code that follows (I did that for st20c2/4 code and have posted that somewhere around here)--which make disassembly of the code quite a bit easier as otherwise you have to manually go through the code and start disassembly everywhere that there is un-disassembled code--might not be that bad with just a boot, but when you have a million bytes to work with the manual method becomes mind numbingly after 4 or 5 hours.

Ida is only as smart as the programmer that wrote it and he is trying to create a program that is good for many processors and not ideal for one particular one. You can get better results from it if you write IDCs for it, but then you only get better results reflecting the skill you put into the programming you do.

Reply With Quote

Page 1 of 4 1 2 3 4 Last

« Previous Thread | Next Thread »

Similar Threads

Apple's EPEAT Withdrawal Raises Recycling, Disassembly Concerns By NEWSpaperBOT in forum PCWorld Latest Tech	Replies: 0 Last Post: 07-10-2012, 08:00 PM
Apple's EPEAT Withdrawal Raises Recycling, Disassembly Concerns By NEWSpaperBOT in forum PCWorld Latest Tech	Replies: 0 Last Post: 07-10-2012, 06:50 PM
CX24155 Dump Disassembly By SabeL in forum Coders Corner	Replies: 7 Last Post: 01-19-2012, 02:48 PM
need help with a sti5107 device based stb By widzo in forum Coders Corner	Replies: 4 Last Post: 07-06-2011, 02:58 PM

Posting Permissions

You may not post new threads	**BB code** is On
You may not post replies	**Smilies** are On
You may not post attachments	**[IMG]** code is On
You may not edit your posts	**[VIDEO]** code is On
	HTML code is Off
	Forum Rules